Efficient key pre-distribution for wireless sensor networks: a case study on 6LoWPAN networks running Contiki-OS

Abstract. The Internet of Things is imposing an evolution of the capabilities of wireless sensor networks. The new IP-based 6LoWPAN standard for low power sensor networks allows an almost seamless connection of local sensor networks to the Internet. This makes it easier to connect local sensor networks to data processing servers or to provide web services for those networks. On the other hand, the connection to the Internet also opens doors for unauthorized nodes to become part of the local network. Therefore, measures need to be taken to protect the information that is captured and communicated within the local network. The most important challenge in doing so, is the implementation of a key management architecture, keeping in mind that the sensor nodes are constrained in power consumption and data storage capacity.
This paper builds on a previously proposed symmetric key management scheme for 6LoWPAN networks. The original scheme is based on wired bootstrapping for the enrollment of new nodes, while the paper at hand proposes a wireless method for key pre-distribution. We analyze the original wired scheme and its shortcomings. Next, we propose the new wireless scheme and elaborate on the practical implementation on Zolertia Z1 nodes running Contiki-OS. We show that it is possible to provide end-to-end security using wireless bootstrapping within the constraints of the tiny nodes at hand.

Keywords: 6LoWPAN, Wireless Sensor Network Security, key pre-distribution, Zolertia Z1, Contiki-OS

1 Introduction

6LoWPAN is a new communication standard for wireless sensor networks [1]. Because the standard is based on IPv6, it allows an easy integration between local sensor networks and the Internet. This way, it is unnecessary to install complex gateways to allow the communication of 6LoWPAN sensor nodes with data processing servers connected to the Internet or to provide web services for the sensor networks. On the downside, connecting local sensor networks to the Internet makes them vulnerable for all known security attacks in IP networks, besides the vulnerabilities that are already present in local wireless sensor networks [2]. Whereas traditional computers in traditional IP networks have a large capacity for the implementation of security architectures, the nodes in wireless sensor networks are limited in data storage and power budget.

In this paper, we present a scheme that provides end-to-end security in 6LoWPAN networks using the Zolertia Z1 hardware platform. We use the open source operating system Contiki [3], which is developed to operate on constrained sensor nodes. Contiki allows multitasking and is C-based. It uses the μIPv6 stack for 6LoWPAN communications. The scheme builds on a previously proposed scheme using wired bootstrapping for the enrollment of new network nodes [4]. We identify the shortcomings of this scheme and propose a new method for key pre-distribution. The new method renders the wired interface obsolete by using the wireless radio for secure bootstrapping. We obtain a higher level of security by pre-configuring the sensor node firmware with a unique secret key that is used during key transfer of the bootstrapping phase. In addition, we augment the user friendliness by allowing multiple nodes to connect at the same time.

The paper is organized as follows. In Sect. 2, we discuss related work and background information. In Sect. 3 we present the proposed key management architecture based on a wireless key pre-distribution mechanism. Next, the implementation and the performance of the scheme are dealt with in Sect. 4 and Sect.5, respectively. Finally, conclusions are drawn in Sect. 6.

2 Background

2.1 Related Work

This paper builds on the scheme that was presented in [4]. In the remainder of the paper, we refer to that scheme as the 'original scheme'. In order to be able to analyze the original scheme and propose a new scheme, this section explains the topology of the original scheme and the wired bootstrapping method.

As depicted in Fig. 1, the key management scheme infrastructure consists of

- a tablet PC,
- a portable Physical Unclonable Function (PUF),
- a trusted central entity (CE),
- a 6LoWPAN edge router (ER),
- the individual sensor nodes.

The original scheme is a distributive scheme that uses parts of the SPINS protocol [5] and the Zigbee 2007 security scheme [6]. It consists of three phases: the bootstrapping phase, the discovery phase and the pairing phase.

In the bootstrapping phase, the tablet and the PUF generate cryptographic strong keys for the entire network. The CE receives these keys through a secure connection and stores them in the database. Then, the ER sends a request to receive a network key over an encrypted tunnel. Sensor nodes are added to the network through a wired interface to the CE. Through the installer laptop, a key request packet is sent over an encrypted tunnel to the central entity, after which the node information is stored in the database. After authorization of the node by the user (through the tablet), the node receives the network key and a unique sensor key.

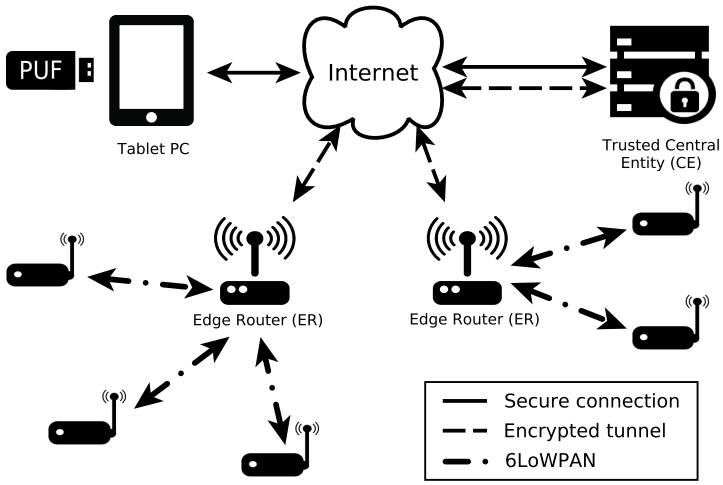

Fig. 1. Topology of the key management scheme

In the discovery phase the sensor nodes use a routing protocol to establish their routes. The communication is secured by the shared network key.

In the pairing phase two entities establish a session key by using a variant of the SNEP protocol [5] based on the Needham-Schroeder symmetric protocol [7]. The CE plays the role of the key distribution center and shares a sensor key with each node in the network. One node sends a request message to another node to initiate the communication. The other node replies with a challenge, after which the first node transfers this challenge, together with additional data on itself, to the CE. After performing the authentication, the CE will send a session key to both nodes, encrypted with their individual sensor keys. To successfully complete the key transfer, the first node demands a verification from the other node, after which the nodes can start communicating in a secure manner.

2.2 Security Goals

Confidentiality protects the overall content or a field in a message. One way to achieve this is by using symmetric key cryptography for encryption with a shared secret key.

Entity Authentication ensures that the other end of a connection or the originator of a packet is the entity that is claimed. Entity authenticity can be obtained by including a message authentication code (MAC) in each packet.

Data Integrity ensures that a packet is not modified during transmission. This is typically achieved by including a message integrity code (MIC) or a checksum in each packet.

Freshness ensures that a malicious entity does not resend previously captured packets.

2.3 Sensor Network Platform

Our selected hardware platform comprises of the Zolertia Z1 [8] sensor nodes running the Contiki OS with the μIP communication stack. The nodes are equipped with a MSP430F2617 microcontroller [9] that offers 8 kB of RAM and 92 kB of Flash memory. The wireless communications are handled by the CC2420 transceiver from Chipcon [10] operating in the 2.4 GHz range. The CC2420 is compliant with the IEEE802.15.4 standard [11] offering MAC and baseband modem support. In addition, it has an AES128 co-processor [12] that operates in three different block cipher modes[1] [13]. During communications the AES core can perform in-line security operations unburdening the load of the microcontroller. Furthermore, it can also be used freely by the application for higher level security operations.

3 Proposed Wireless Key Pre-distribution Mechanism

In this section, we present our wireless key pre-distribution mechanism for efficient security bootstrapping of wireless sensor nodes. This work aims to improve the user friendliness and security trade-offs for the wired bootstrapping mechanism described in [4]. We start with an analysis of the wired bootstrapping system in Sect. 3.1. Followed by the general structure of our solution in Sect. 3.2 and finally, the security properties in Sect. 3.3.

3.1 Analysis of The Wired Bootstrapping Mechanism

Starting from the symmetric key management scheme described in Sect. 2.1, we notice that key pre-distribution is required to provide the sensor nodes with network associated key material. Without the appropriate key material, the nodes are not able to participate in secure network communications. Furthermore, loading the keys into the nodes prior to deployment is not possible due to absence of network association. The original scheme solves this problem by using a wired bootstrapping mechanism to initialize the nodes with the necessary security information during node placement [4]. After a thorough study we noticed that the described method comes with some disadvantages.

Scalability The wired interface has a negative influence on network scalability. When facing larger networks the key bootstrapping becomes tedious and unmanageable due to the limitation of configuring one node at a time.

Applicability The need for an additional communication channel on the nodes limits the applicability of the entire symmetric scheme. Not only does it require additional resources to implement the interface, nodes without a serial interface are not able to participate in secure communications.

[1] Three block cipher modes are: Cipher-block chaining mode (CBC-MAC), Counter mode (CTR) and Counter with CBC-MAC (CCM)

Security Due to the unencrypted key transfer between the node and its access point, the scheme is vulnerable for eavesdropping attacks [2]. Furthermore, adding nodes requires the trust of additional hardware for secure key transfer. Another threat arises when attackers start using the wired interface as a device access point. Therefore, proper precautions must be taken into account when implementing the interface.

3.2 General Structure of the Proposed Solution

After evaluating the wired bootstrapping mechanism we opted for a wireless approach for key pre-distribution. The general structure of our solution can be divided into two separate preparation phases. The first phase is handled by the sensor node vendor and the second by the user during deployment.

Vendor Preparation
The sensor node vendor supplies the user with the sensor node hardware and firmware. In our solution, the vendor is required to extend the firmware with an unique pre-installed Bootstrap Key (Kb) that is linked to the node's MAC address or node ID. The Kb's are used only temporary during the "user preparation" phase for secure key transfer between the Central Entity (CE) and the sensor node. Every Kb is derived from a Master Boostrap Key (Kmb)[2] using a Key Derivation Function (KDF) such as [14] in combination with a node's MAC address. By using the node ID as additional KDF input we ensure that each Kb is unique and related to the specified sensor node. These firmware preparations are illustrated in Fig. 2.

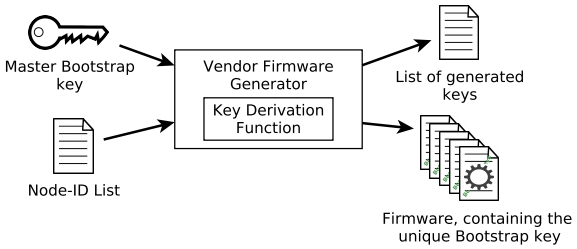

Fig. 2. Vendor firmware preparation

Providing the KDF with the Kmb and a list of all the node IDs, the vendor firmware generator is able to create firmware files containing an unique Kb. This results in an unique firmware file for each sensor node. Moreover, the vendor collects all the Kb's with their associated node ID in a list that is used during the "user preparation" phase.

[2] The Master Boodstrap Key is held secret by the vendor to prevent attackers from cloning the generated Bootstrap Keys.

User Preparation
After the "vendor preparations" the nodes containing the required firmware are ready for deployment. The user starts by updating the CE security database with the list of generated Kb's from the vendor. This is done by uploading the list into the tablet and transferring the keys to the security database through a secure connection as is shown in Fig. 3. Once the database is up-to-date the CE is ready to accept key requests from the sensor nodes.

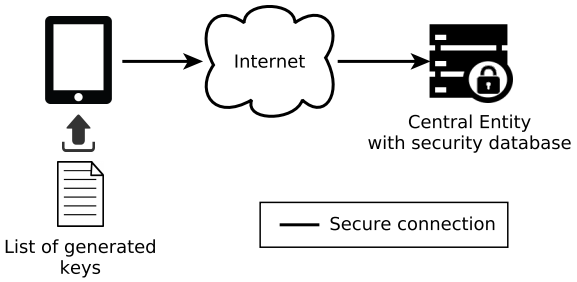

Fig. 3. Central Entity (CE) database preparation

If a node wants to join a network it sends out a key request packet through its wireless radio. The packet is unencrypted and contains information about the node. The user has to ensure that, during the key request, the nodes are within a range of 10 centimeters of the 6LoWPAN edge router (ER). This allows the ER to ignore unencrypted packets from which the signal strength is below a predefined value. Other initialized nodes in the network will automatically drop the unencrypted packets by default. Fig. 4 illustrates the various components of the setup.

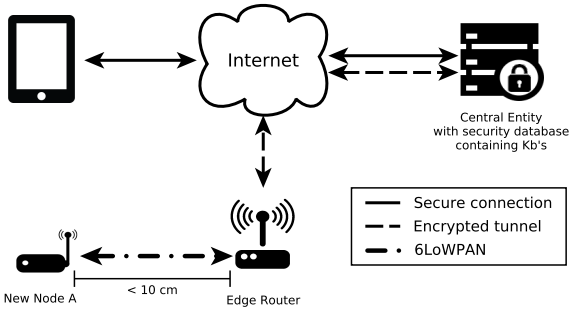

Fig. 4. Adding new nodes

Once the ER receives and identifies a valid key request, it is transfered through an encrypted tunnel to the CE for further processing. The CE anal-

yses the node information from the key request packet and checks whether the node ID is linked to a valid Kb in the security database. Upon a valid check the request is stored and awaits approval from the user through the tablet. Otherwise, the key request is dropped. In case multiple key requests from the same node arrive, only the first one is stored.

The next step requires the user to manually approve the stored key requests. Similar to the original scheme the user makes use of the tablet interface to either acknowledge or decline a request [4]. For each approved request the CE creates a unique security device in its database which contains all the security information[3] needed by the requesting node. This information is encrypted using a block cipher such as AES [12] by the CE using the individual Kb. After completion a reply packet is created containing the encrypted message and sent back to the ER for broadcasting in the 6LoWPAN network.

Finally, the requesting node receives a reply from the CE through the ER and tries to decrypt the packet with its own Kb. Upon successful decryption, the security information is stored and the node is ready to participate in network communications.

3.3 Wireless Bootstrapping Properties

We evaluated our solution using the same analysis as in Sect. 3.1.

Scalability In contrast to the wired bootstrapping mechanism, our system has good scaling properties. Due to the absence of a physical connection the nodes are able to request keys in parallel, greatly reducing the deployment time.

Applicability By using the wireless radio as bootstrapping channel, we allow devices without a wired interface to participate in secure network communications. In addition, we save node resources by reusing existing components in the sensor nodes.

Security Our solution provides confidentiality, integrity and authenticity during key transfer as well as replay protection by incorporating a nonce counter. In comparison to the wired bootstrapping, we eliminate the need to trust additional hardware operated by a third party.

Robustness The system has good resilience against node capture. Consider a scenario where multiple nodes are physically compromised and the Kb's are extracted. Other nodes are not affected by the compromise because every node has an unique Kb. Furthermore, revoking Kb's is trivial due to the centralized topology of the network. On the other hand, obtaining Kmb with information from the individual Kb's should be very difficult because of the KDF.

[3] Security information contains key material that is linked to the network where the key request was registered.

4 Implementation of The Wireless Bootstrapping Mechanism

The implementation of the wireless bootstrapping mechanism is spread out over various components. We start by discussing the vendor firmware generator in Sect. 4.1. Consecutively, we describe the Contiki implementation on the individual sensor nodes in Sect. 4.2 and the extension of the wireless access points in Sect. 4.3. Finally, we describe the CE server configuration in Sect. 4.4.

4.1 Vendor Firmware Generator

As mentioned in Sect. 3.2, the vendor is responsible for providing the sensor nodes with unique firmware files containing the Kb. In addition, the vendor should provide a list of key value pairs linking the node ID to a matching Kb.

We use the standardized KDF1 key derivation function [14] in combination with the NIST recommended SHA3-256 hashing algorithm [15]. In order to reduce implementation time we use the open source Java Bouncy Castle crypto library to provide us with an implementation of the SHA3 algorithm [16].

With the KDF operational we used scripting to create the vendor firmware generator. The script derives a Kb from the specified Kmb using a copy of the node IDs as additional input for the KDF. Each generated Kb is then copied to a placeholder in one of the sensor node's source files and compiled to an unique firmware file as illustrated in Fig. 5. Besides firmware generation, the script also creates an XML file containing a list of the node IDs and their derived Kb's.

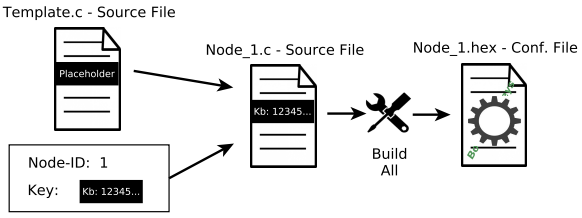

Fig. 5. Vendor firmware generator operation

4.2 Sensor Node Contiki Implementation

The wireless bootstrapping implementation on the sensor nodes is focused on area and robustness. We tried to use as many built-in components of Contiki to reduce the memory consumption as well as the complexity of our implementation, e.g. MAC broadcasting. The bootstrapping functionality is mainly situated in the MAC layer of the μIP stack and makes use of its broadcast capabilities. The next paragraph describes the node's behavior when connecting to a network.

Consider a blank sensor node without associated key material wanting to join a network. Starting with a system reset the node checks its memory for key material. When dealing with a blank node, this check fails and the node initiates the bootstrap procedure. Immediately after initialization, the node sends out a single unencrypted hello message containing the node ID and a one-byte message code indicating a key request packet. Because the node has no knowledge of the surrounding entities or the CE, the packet is broadcasted on MAC level using the existing Contiki functions. Once the hello message has been transmitted the node waits for a reply message coming from the CE. Note that during the bootstrapping state the node is not able to react on any packets including routing messages except hello reply messages.

When receiving a packet of the correct length the node considers the message to be a hello reply coming from the CE. The hello reply message is an encrypted packet containing the requesting node's key material. The packet is encrypted using the AES block cipher in combination with the node's Kb, providing packet confidentiality and data integrity as well as authenticity and packet freshness. The integrity is preserved by calculating and appending a MIC tag of 8 bytes to the message. On the other hand, authenticity is provided by the fact that only two entities have knowledge over the Kb, being the node in question and the CE. The packet freshness is implemented by incorporating a nonce counter in the message. The reply message format is shown in Fig. 6.

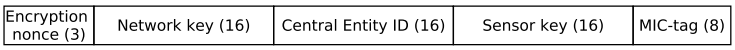

Fig. 6. Hello reply message format (size in Bytes)

The node tries to decrypt the reply packet using its Kb as decryption key for the dedicated AES core in CCM mode. In case the message authentication fails, the node drops the packet and returns to its wait state. Otherwise, the packet is parsed and the key material is extracted and stored in the node's memory. Once the keys are stored, the program performs a software reset forcing re-initialization. This time the node detects the key material and normal operations are started. Note that the Kb is only used during key transfer of the wireless bootstrapping, not during normal security operations.

4.3 Wireless Access Point Contiki Implementation

The wireless bootstrapping mechanism is implemented as an extension of the wireless access point or Edge router (ER) functionality. In contrast to the wireless sensor nodes, the ER has to cope with both normal security operations and the wireless bootstrapping operations at the same time. Therefore, we chose to extend the Contiki process on the ER in order to handle these additional requirements. The implementation resides in the MAC layer and application layer of the μIP stack. We start by explaining the ER behavior during key requests

followed by the behavior during hello reply messages.

Starting with the MAC layer, as described in the previous Sect. 4.2, the nodes send out an unencrypted key request packet as a MAC broadcast message. In normal circumstances these packets would be ignored by the ER due to failed integrity check at MAC level. However, by verifying the signal strength and the message length the ER allows these packets to flow through for further processing. In our case we chose a minimum signal strength value that translates in a distance of approximately 10 centimeters.

The unencrypted packets that flow through are considered to be key request messages and are handed over to the ER process. The ER process resides at the application layer and has access to a UDP connection. Since the nodes do not know the CE's address, the ER process is responsible for forwarding the packet to the CE. This is done by extracting the key request payload and formating it to a UDP packet designated for the CE. Upon completion, the UDP packet is transmitted through an encrypted tunnel to the CE.

Once the key request has been processed a reply is generated by the CE. The hello reply is forwarded over the UDP connection, through the encrypted tunnel, back to the ER. The ER process translates the reply message payload to a MAC broadcast packet and hands it over to the wireless radio for transmission. Note that the requesting node should be in a one-hop radius from the ER to receive the reply packet.

4.4 Central Entity Server Configuration

Since the Central Entity holds the cryptographic key material of the entire network, additional care must be applied when extending its functionality. Careless implementations can lead to vulnerabilities which can be exploited by adversaries. Therefore, we focused on providing a robust implementation of the CE's key pre-distribution mechanism. The complete mechanism is written as a standalone Java UDP server that listens to incoming packets coming from the encrypted tunnels. The server stands in direct contact with the security database through a secure connection, giving access to the necessary key material.

In case of an incoming key request, the packet is parsed and followed by an immediate identity check. The identity of the node can be verified by the presence of a Kb record in the database. Upon an invalid result the packet is dropped. Otherwise, the server continues with verifying if the request was already processed before storing it in the database. Once the request is stored, it awaits approval from the user, as described in Sect. 3.2.

A separate thread, that makes use of the same UDP connection, handles the approved requests. For each request, the server creates a new security device in its database and selects new key material according to the origin of the request. The origin is verified by checking the address of the ER where the request was

registered. Finally, a reply packet is created and encrypted using the corresponding Kb as encryption key for the AES-CCM core. We used the Bouncy Castle implementation of AES-CCM for encrypting the packets.

5 Performance Analysis

One of the most important aspects of wireless sensor networks is resource management. Due to the limited availability of processing power, memory and energy it becomes difficult to implement complex functionality on the sensor nodes. For that reason we compared the memory requirements of our implementation with the requirements of the wired bootstrapping interface. The results are shown in Fig. 7. The measurements are normalized to the total available memory of the Zolertia Z1 platform, which is 92 kB of ROM and 8 kB of RAM memory.

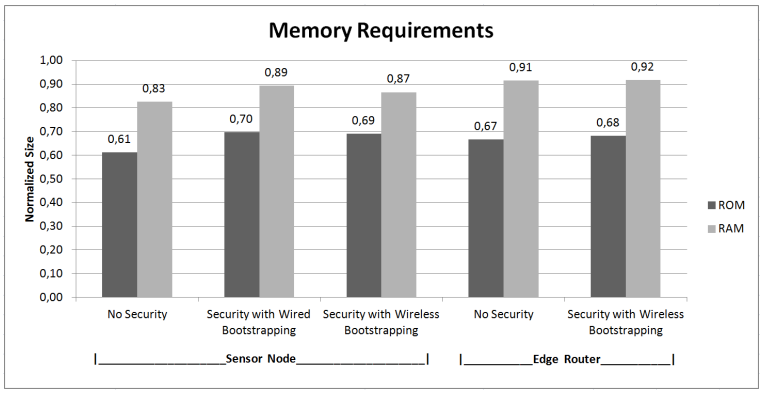

Fig. 7. Memory requirements of the different Contiki implementations on both the sensor node and ER

The sensor node measurements are given for a UDP client example program. First of all, we list the UDP client implementation without security as a reference for the memory requirements of security with bootstrapping. Then we give the results of the wired bootstrapping implementation, followed by the wireless bootstrapping. Note that the differences in memory requirements between the wired and wireless implementations are fairly minimal. However, if we consider the gains in user friendliness, the freed-up resources as explained in Sect. 3.1, and the stronger security of the wireless bootstrapping mechanism; our wireless solution is more favorable.

The memory requirements of the edge router extension are also shown in Fig. 7. Here we see that extending the functionality with the wireless bootstrapping mechanism has a small impact on memory requirements.

It is pointed out that the results of the vendor firmware generator and the CE java server are not included in Fig. 7, because they are irrelevant in this comparison.

6 Conclusion

We presented our wireless bootstrapping mechanism as an improvement on the wired bootstrapping mechanism described by Smeets et al. [4]. We aim at achieving both higher security and increased user friendliness without the need for additional hardware. Our solution is integrated in the Contiki OS and evaluated on the Zolertia Z1 platform with the focus on a robust implementation. The results show that by using wireless bootstrapping we achieve reduced memory costs in comparison to the wired mechanism with the advent of stronger security. In the future we plan to examine the effect of various attacks against the wireless bootstrapping mechanism and we also plan to increase the resilience against Denial-Of-Service attacks.

References

1. Shelby, Z., Bormann, C.: 6LoWPAN: The Wireless Embedded Internet. Published by John Wiley & Sons Ltd., (2009)
2. Çayırcı, E., Rong, C.: Security in Wireless Ad Hoc and Sensor Networks. Published by John Wiley & Sons Ltd., (2009), 107–283.
3. Dunkels A., Gronvall B., Voigt T.: Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: Proc. of the 29th Annual IEEE International Conf. on Local Computer Networks (IEEE Computer Society), Washington (2004).
4. Smeets R., Aerts K., Mentens N., Singelee D., Breaken A., Segers L., Touhafi A., Steenhaut K., Niccolo D.: A cryptographic key management architecture for dynamic 6LowPan networks. In: Proc. of the 9th International Conf. on Applied Informatics (ICAI), (2014)
5. Perrig A., Szewczyk R., Tygar J.D., Wen V., Culler D.E.: Spins: Security protocols for sensor networks. Wirel. Netw. 8(5), (2002), 521–534.
6. E. Yksel, H.R. Nielson, and F. Nielson: ZigBee-2007 Security Essentials. In: Proc. of the 13th Nordic Workshop on Secure IT Systems (NordSec), (2008)
7. Needham RM, Schroeder MD: Using encryption for authentication in large networks of computers. Communications of the ACM 21 (12), (1978), 993–999
8. Zolertia: Zolertia Z1 datasheet, `http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf`
9. Texas Instruments: MSP430F2617 datasheet, `http://www.ti.com/lit/ds/symlink/msp430f2617.pdf`
10. Chipcon: CC2420 datasheet, `http://www.ti.com/lit/ds/symlink/cc2420.pdf`
11. IEEE 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks. USA: IEEE (2003)
12. Daemen J., Rijmen V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag, (2002)
13. National Institute Of Standards and Technology: Recommendation for Block Cipher Modes of Operation. (SP800-38A), March (2012)
14. ISO/IEC 18033-2:2006: Information technology - Security techniques - Encryption algorithms - Part 2: Asymmetric ciphers. Ed. Victor Shoup, (2006)
15. Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: The Keccak SHA-3 submission, `http://keccak.noekeon.org/Keccak-submission-3.pdf`, (2011)
16. The Legion of the Bouncy Castle: Cryptography APIs, `https://www.bouncycastle.org`